GRAPH PROCESSING: A KILLER-APP FOR PERFORMANCE MODELING

Ana Lucia Varbanescu[1,3]

Work with Merijn Verstraaten[1,2], Dante Niewenhuis[1], Ahmed Musaafir[1]

1 UNIVERSITY OF AMSTERDAM

2 netherlands eScience center

3 UNIVERSITY OF TWENTE.

Graph processing ...

… is / can be / will be everywhere![1,2]

- Social networks
- Bioinformatics
- Pandemic analysis[3]
- Fraud detection
- Neural networks
- …

¯_(ツ)_/¯

What about performance?

[1] Sherif Sakr et al.
“The Future Is Big Graphs: A Community View on Graph Processing Systems” – CACM Sept. 2021
[2] Tim Hegeman, Alexandru Iosup
“Survey of Graph Analysis Applications” - arXiv:1807.00382
[3] https://neo4j.com/graphs4good/covid-19/

Large Scale Graph Processing

- Graph processing is (very) data-intensive
  - 10x larger graph => 100x or 1000x slower processing
- Graph processing becomes (more) compute-intensive
  - More complex queries => ?x slower processing
- Graph processing is (very) dataset-dependent
  - Unfriendly graphs => ?x slower processing

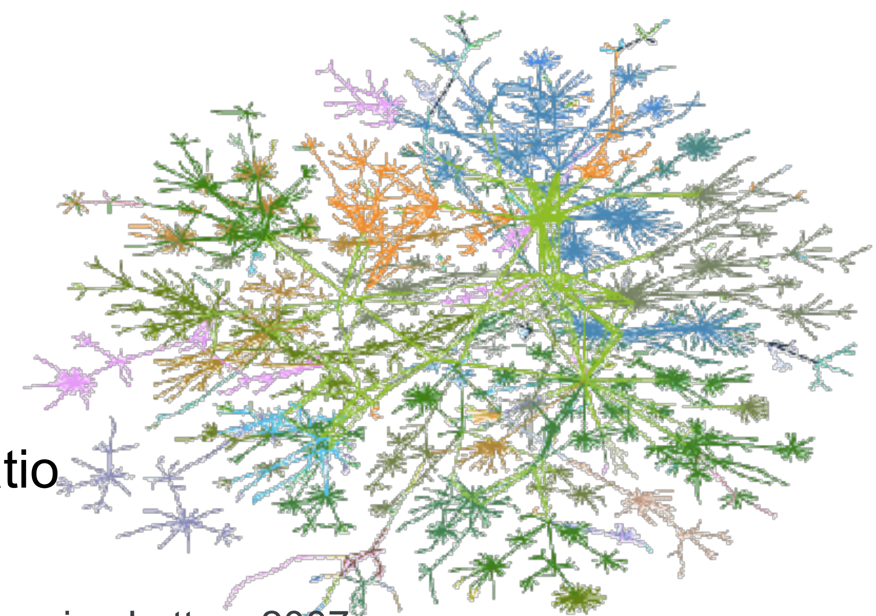
We need parallel algorithms & architectures to enable *more complex analytics* on *larger graphs.*

Parallel graph processing

- Current *PUs
  - Massive (data) parallelism
  - Optimized for high throughput processing
  - Penalties for irregular execution
  - Penalties for load imbalance

(mis)match?

- Graph processing [4]
  - Data-driven computations
  - Irregular memory accesses
    - Poor data locality
  - Unstructured problems
  - Low computation-to-data access ratio



[4] Andrew Lumsdaine et al.
"Challenges in Parallel Graph Processing" – Parallel Processing Letters 2007

Parallel graph processing

- Current *PUs
  - Massive (data) parallelism
  - Optimized for high throughput processing
  - Penalties for irregular execution
  - Penalties for load imbalance

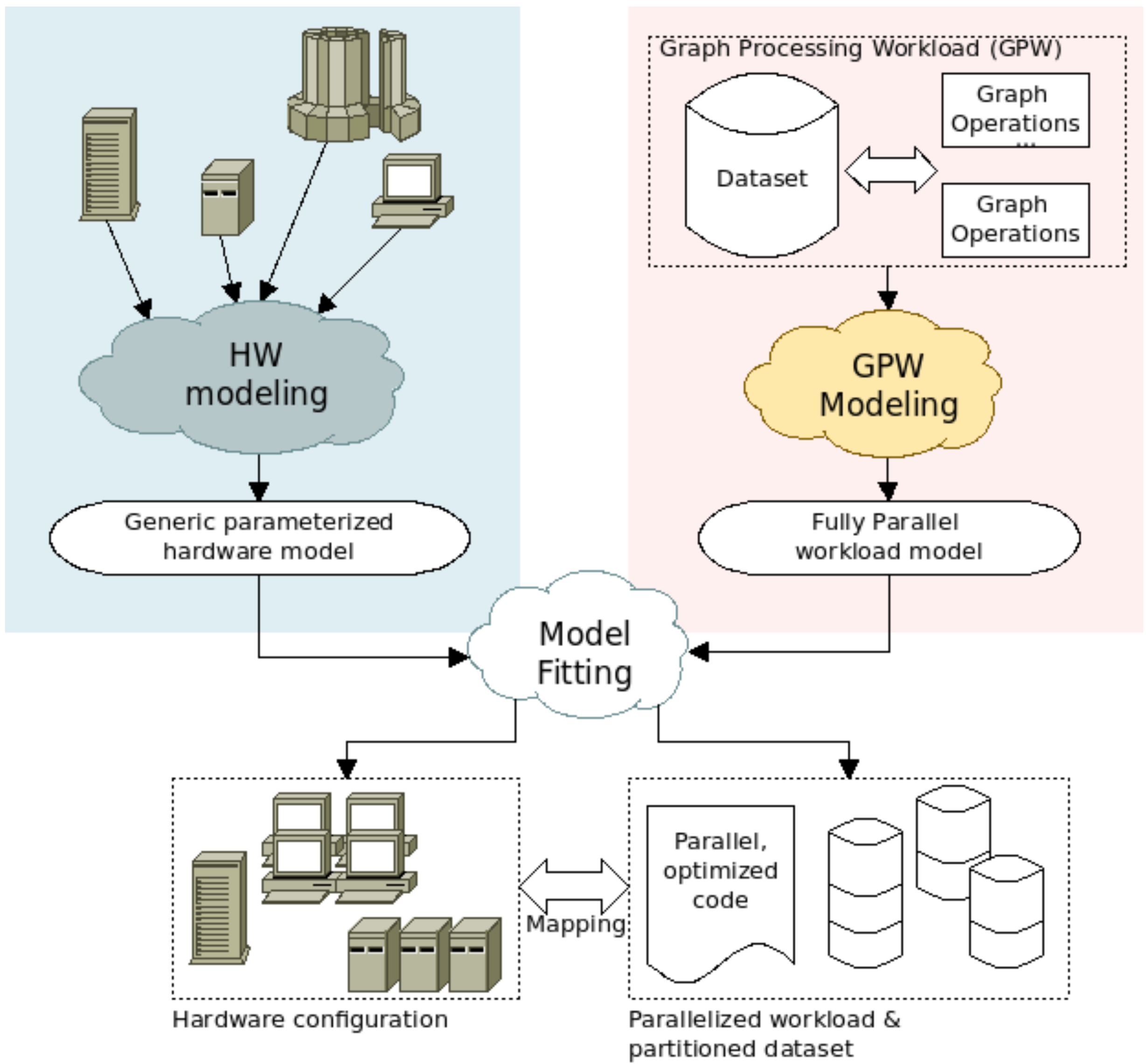
(mis)match?

- Graph processing [4]
  - Unstructured problems
  - Low computation-to-data access ratio

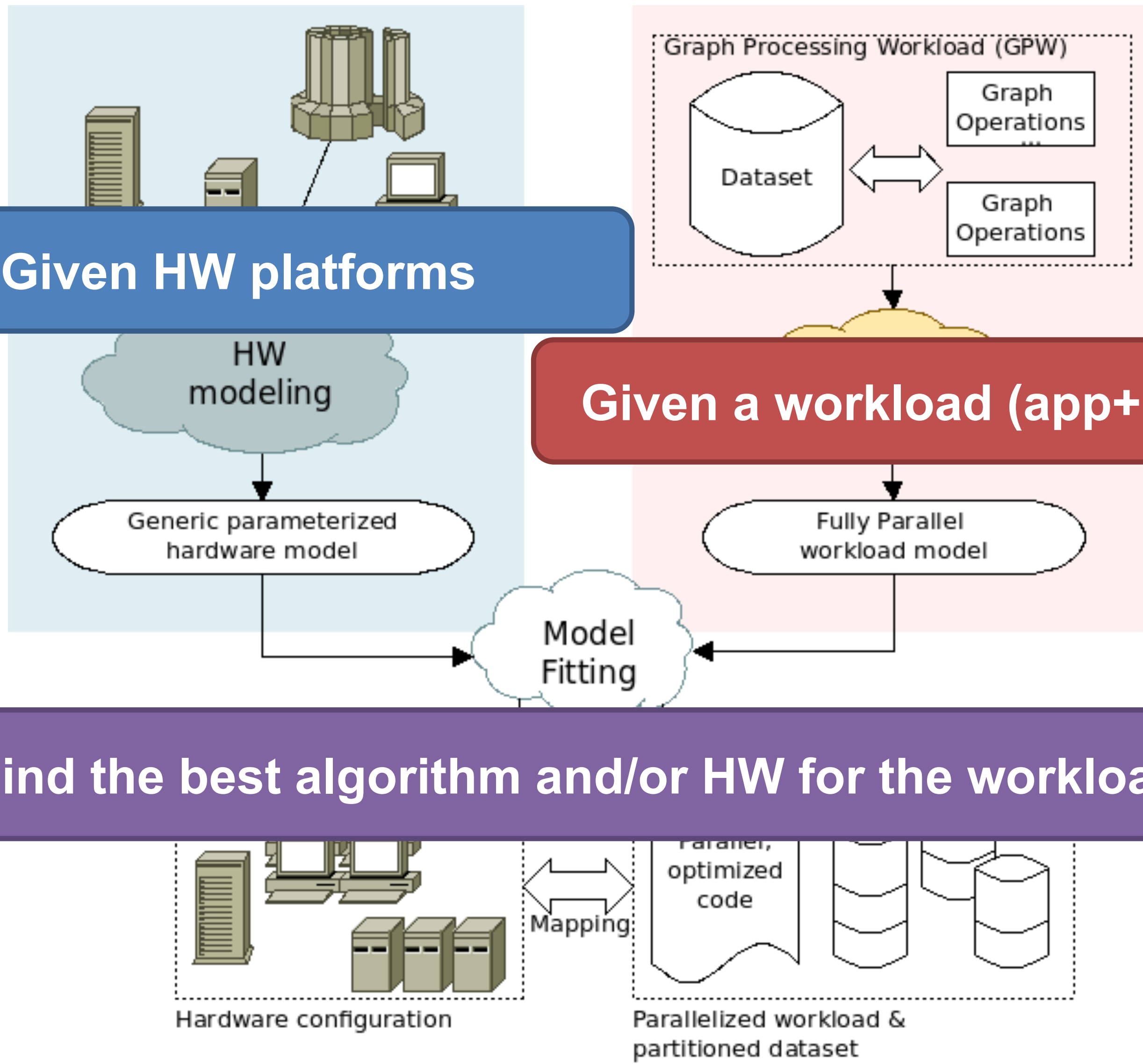
Parallelism <=> New algorithms, data-structures, and graph processing systems

[4] Andrew Lumsdaine et al.
"Challenges in Parallel Graph Processing" – Parallel Processing Letters 2007

Goal: Efficiency by design



Goal: Efficiency by design



Given HW platforms

Given a workload (app+data)

Find the best algorithm and/or HW for the workload

Today's headlines

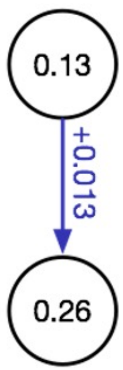
1. ~~Motivation & Challenges~~
2. Performance analysis
   … or how bad can it be?!
3. Controlled experiments
   Graph generators
4. Performance prediction
   Analytical models
   Machine-learning
5. The best BFS algorithm
6. Improving SCC
   Machine learning, again
7. Take home message

"Larry, do you remember where we buried our hidden agenda?"

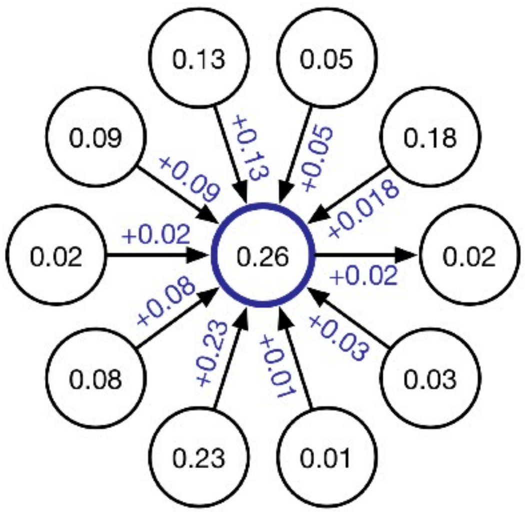
2. Performance analysis

Neighbour iteration

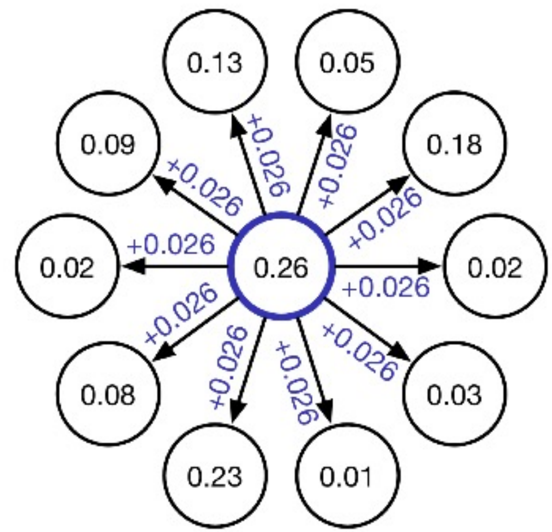
- Various implementations



Edge-centric



Vertex-centric, pull-based



Vertex-centric, push-based

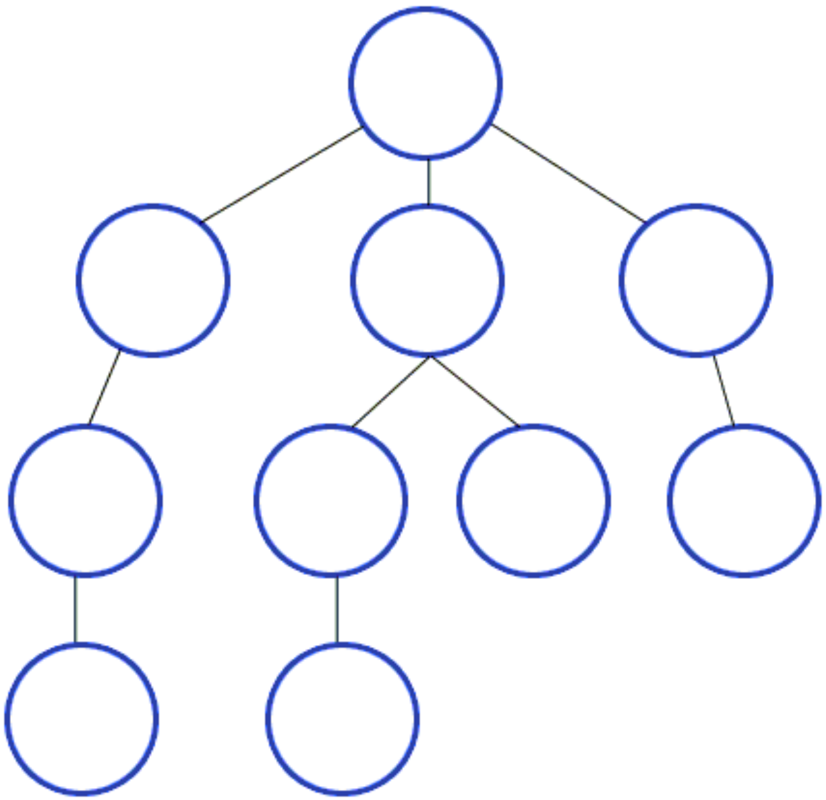
Performance analysis

- NVIDIA TitanX + CUDA 10.0
- Results presented on 9 graphs (ID 1-9 in following fig's)

Id	Graph	# Vertices	# Edges	Dataset
1	actor-collaboration	382,219	30,076,200	KONECT
2	amazon0601	403,394	3,387,390	KONECT
3	flixster	2,523,390	15,837,600	KONECT
4	jester1	73,512	8,272,720	KONECT
5	patentcite	3,774,770	16,518,900	KONECT
6	wikipedia_link_en	12,151,000	378,142,000	KONECT
7	wiki_talk_ru	457,017	919,790	KONECT
8	higgs-social_network	456,626	14,855,800	SNAP
9	sx-stackoverflow-c2q	1,655,350	11,226,800	SNAP

BFS traversal

- Traverses the graph layer by layer
  - Starting from a given node
- Sensitive to …
  - High diameter
  - Graph density
  - (dis)connected components
  - …
- Challenges
  - No computation
  - Load-balancing
  - Irregular memory accesses

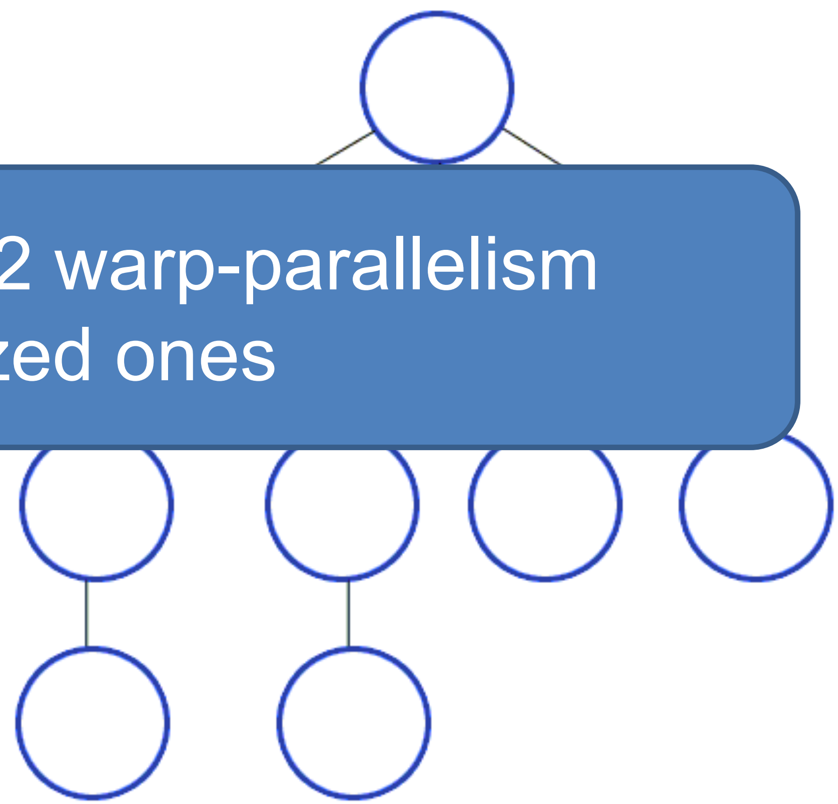


BFS traversal

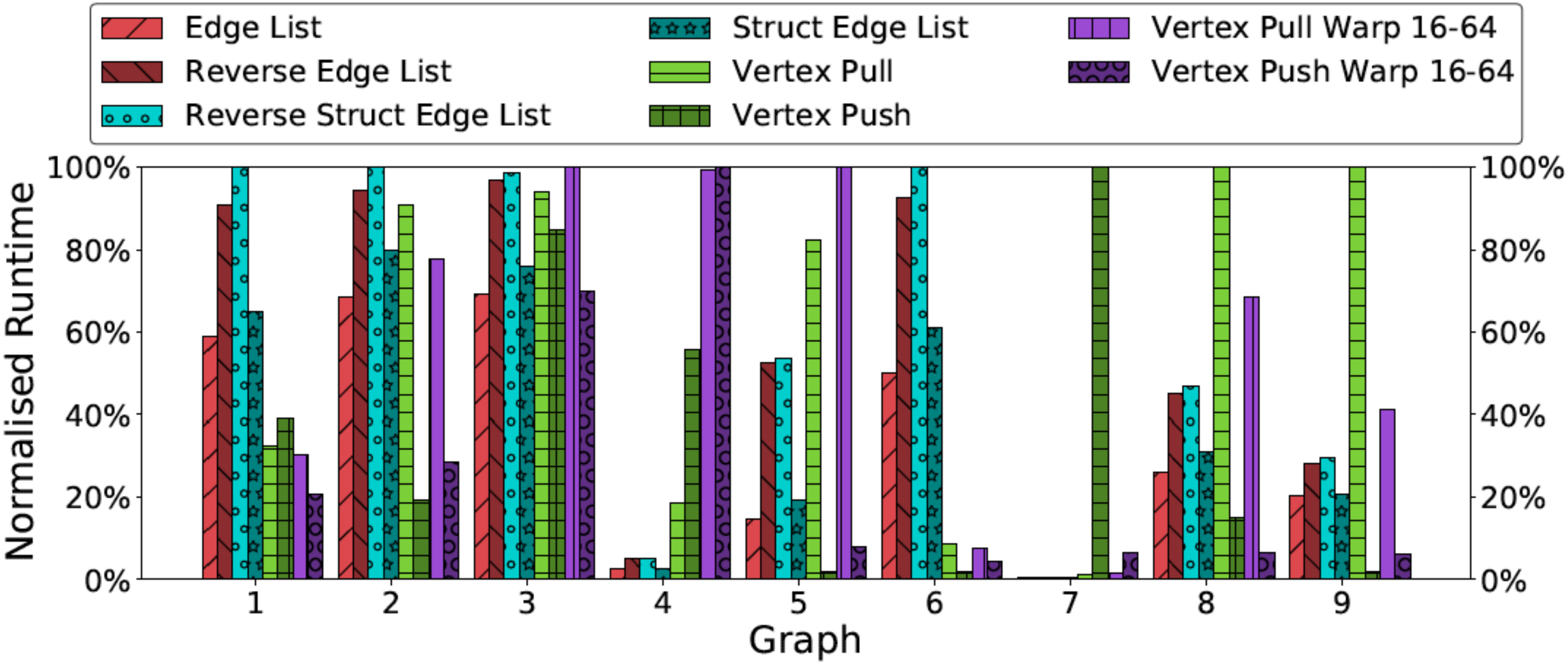
- Traverses the graph layer by layer
  - Starting from a given node
- Sensitive to …
  - High diameter
  - Graph density

We use 6 versions + 2 warp-parallelism parameterized ones

  - No computation
  - Load-balancing
  - Irregular memory accesses

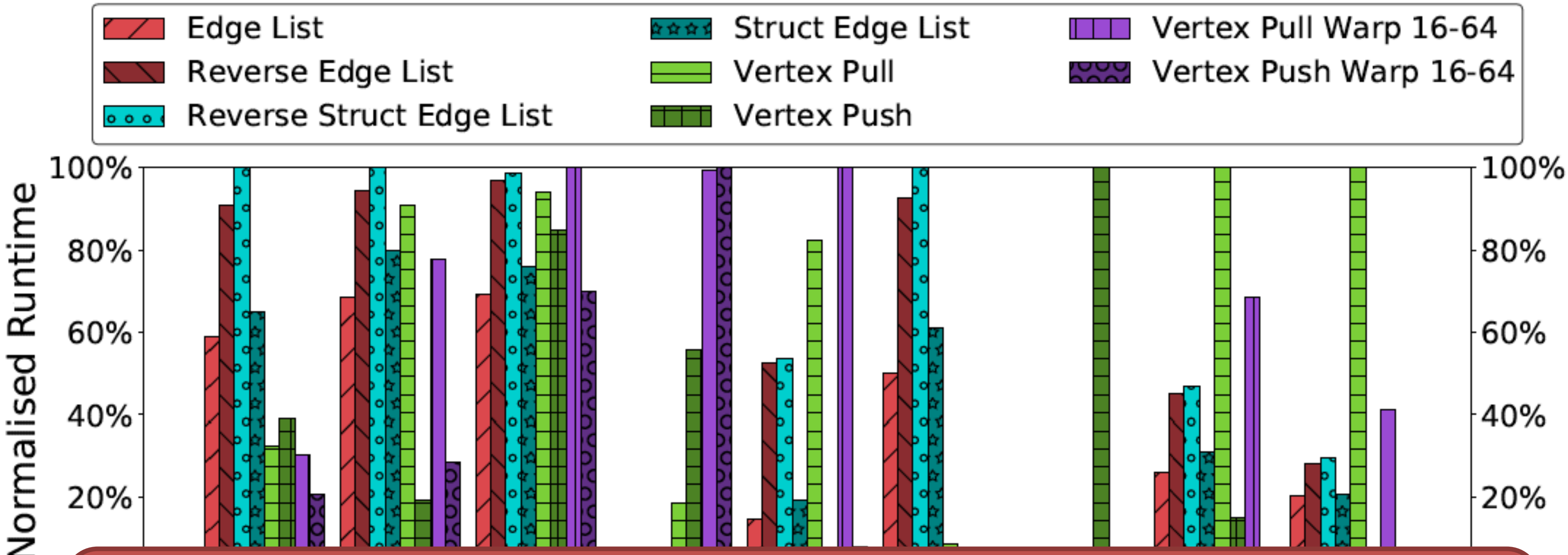


BFS: results



https://github.com/merijn/Belewitte

BFS: results



- Different algorithms behave best.
- Different algorithms behave worst.
- The gap in execution time can be up to 2 orders of magnitude.

Choosing the right / wrong algorithm can really make a difference!

PageRank calculation

- Calculates the PR value for all vertices
  - Assign value to each vertex
  - Repeat until convergence
    - Collect PR for all incoming edges
    - Update vertex PR
- Sensitive to …
  - Graph density
  - Degree distribution
  - ”sink” nodes
- Challenges
  - No computation
  - Load-balancing
  - Irregular memory accesses

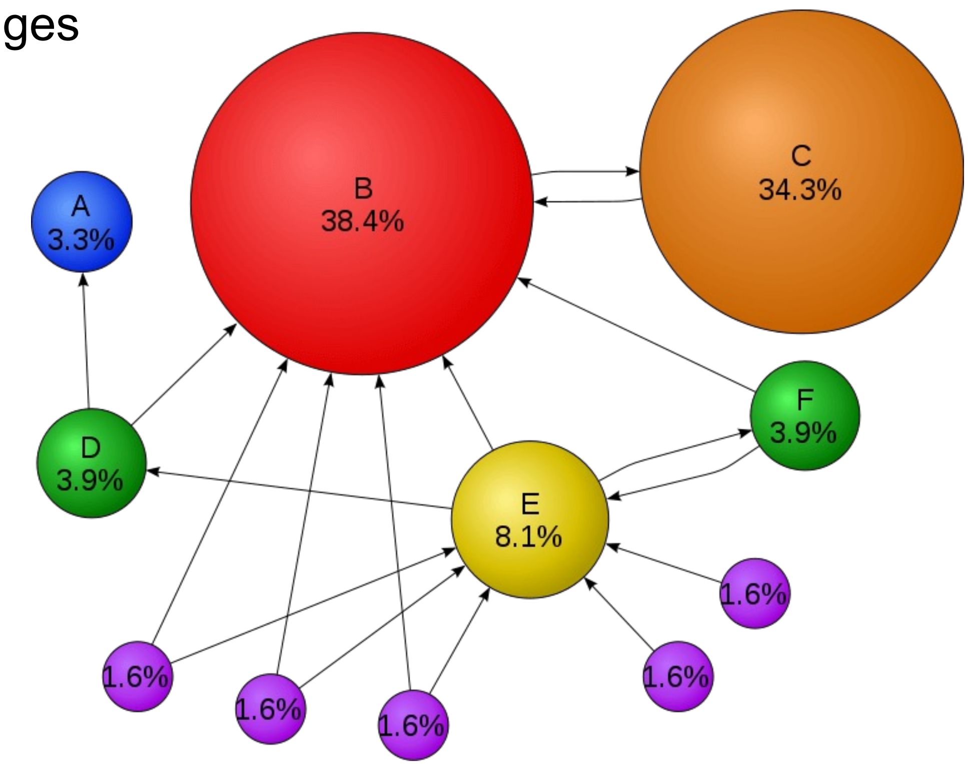


Image courtesy of: https://en.wikipedia.org/wiki/PageRank

PageRank calculation

- Calculates the PR value for all vertices
  - Assign value to each vertex
  - Repeat until convergence
    - Collect PR for all incoming edges
    - Update vertex PR

We use 7 versions + 2 warp-parallelism parameterized ones

- Challenges
  - No computation
  - Load-balancing
  - Irregular memory accesses

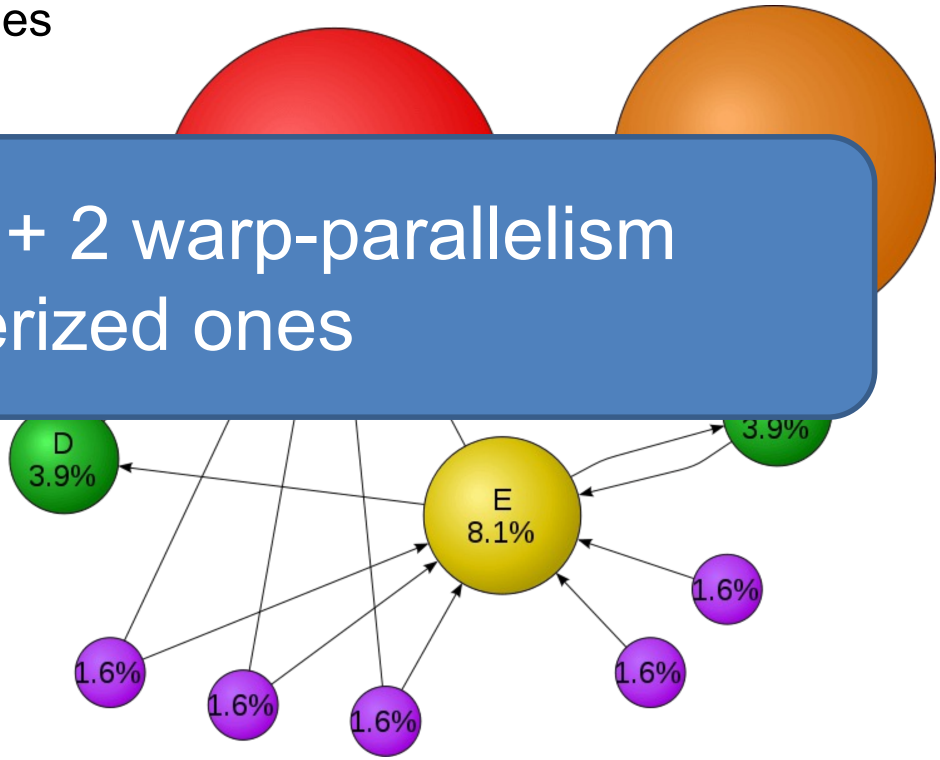
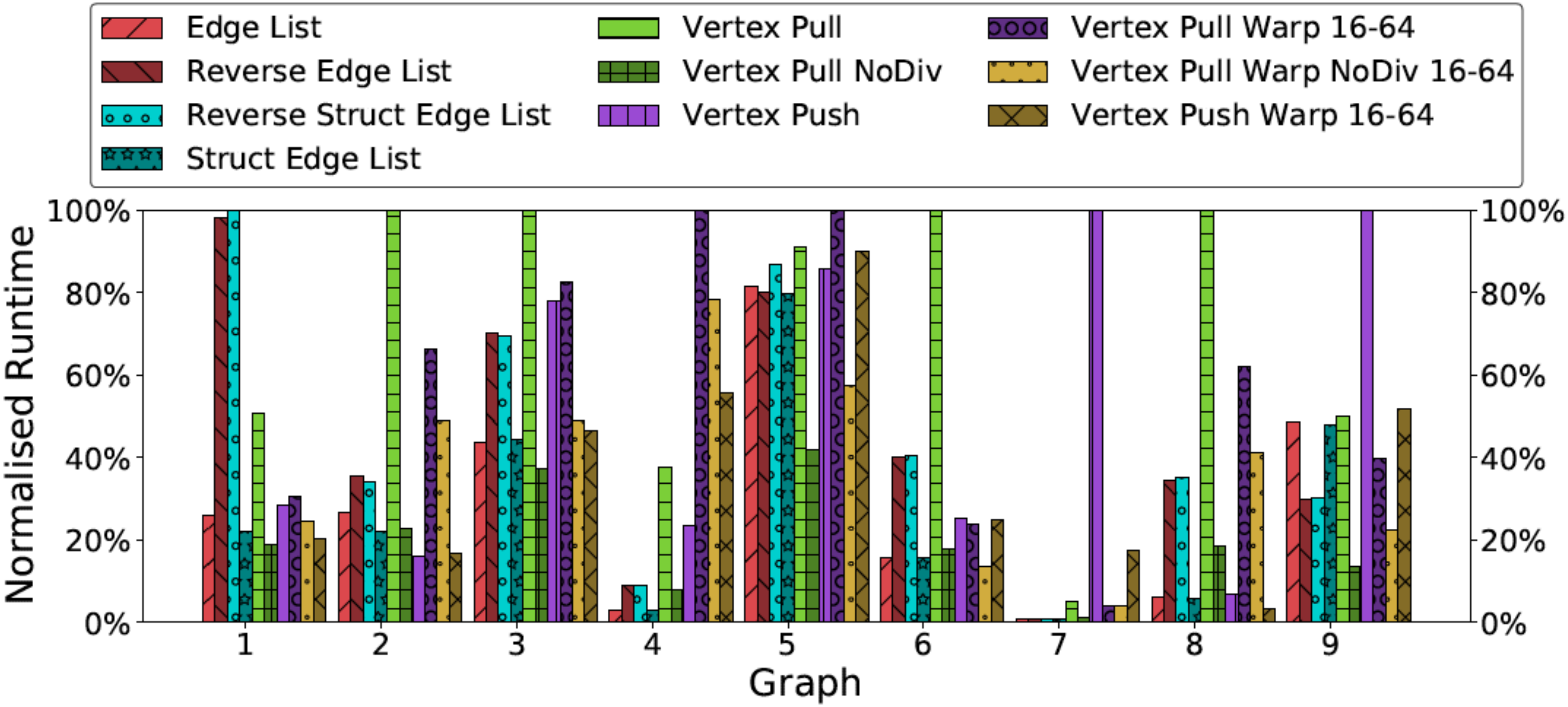


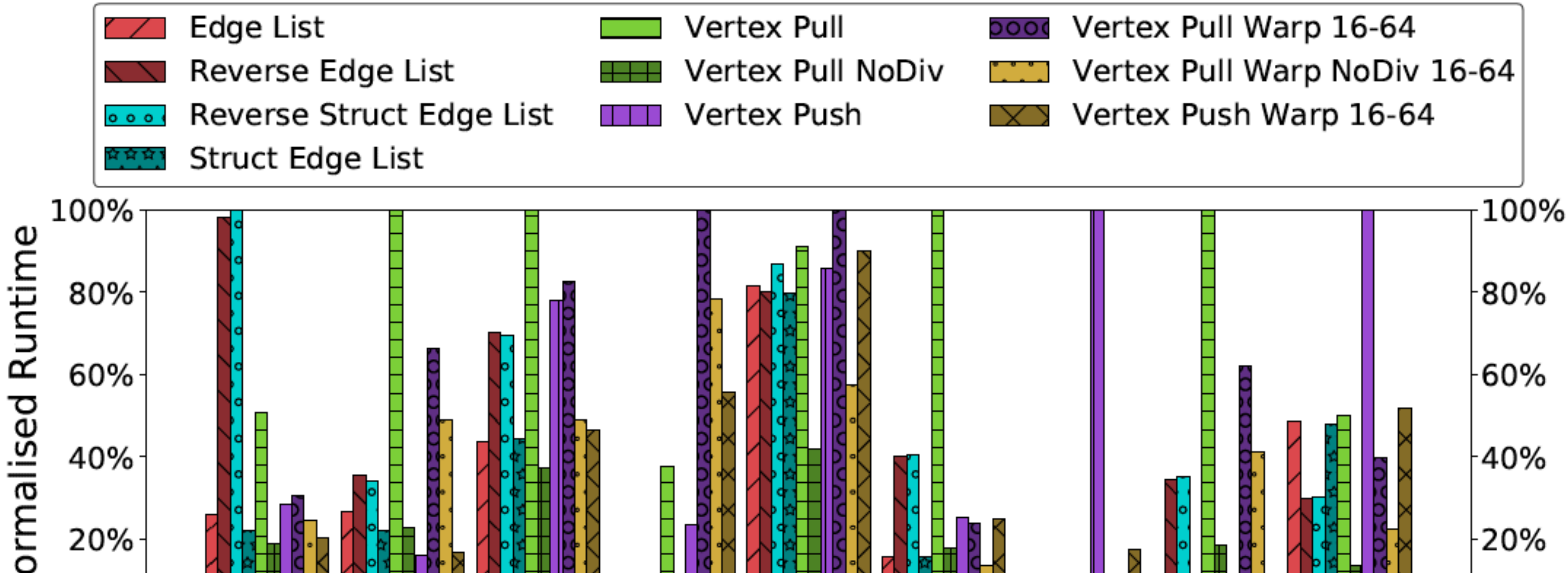
Image courtesy of: https://en.wikipedia.org/wiki/PageRank

PageRank: results



https://github.com/merijn/Belewitte

PageRank: results



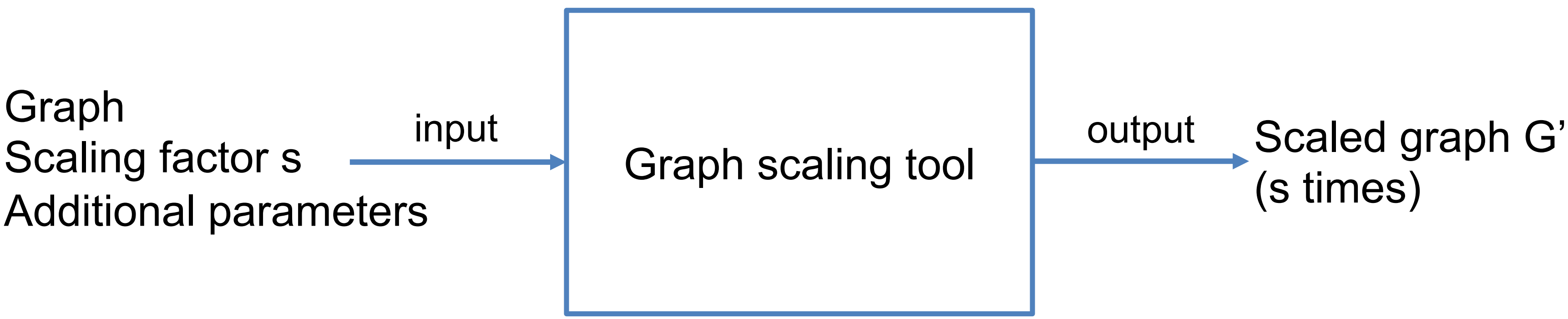
- Different algorithms behave best.
- Different algorithms behave worst.
- The gap in execution time can be up to 2 orders of magnitude.

Choosing the right / wrong algorithm can really make a difference!

3. Controlled experiments

Graph “scaling”*

- Generate “similar” graphs of different scales
  - Control certain properties

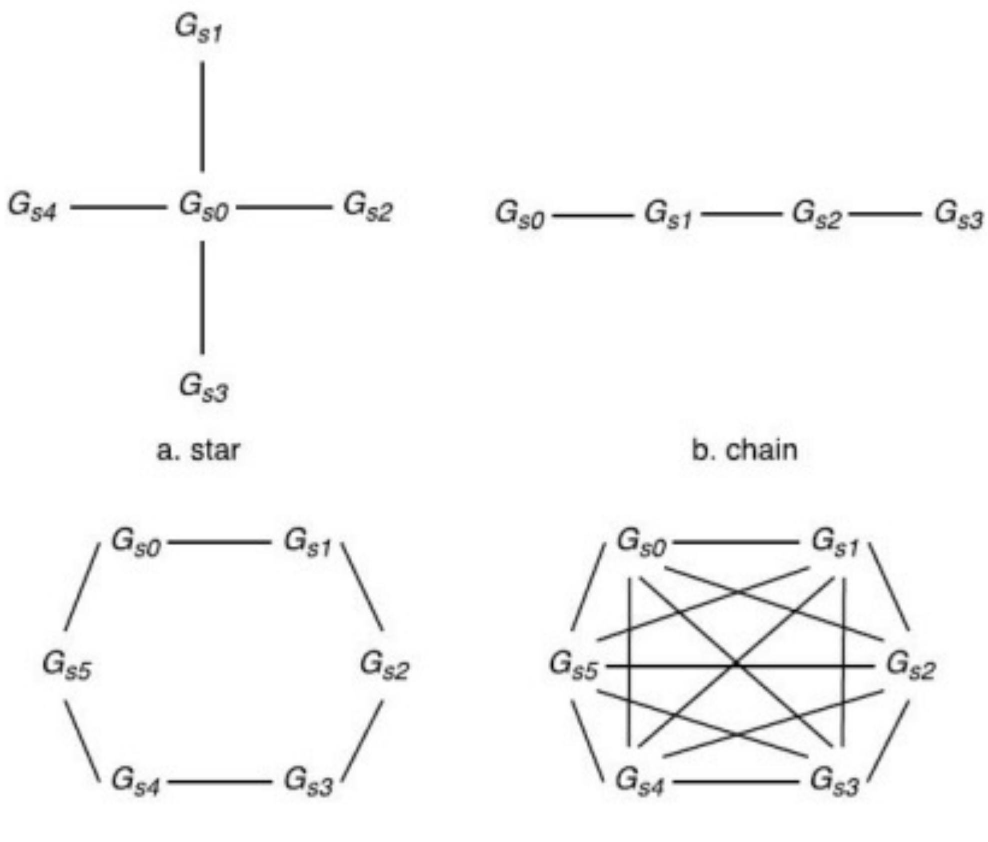


*A. Musaafir et.al – “[A Sampling-Based Tool for Scaling Graph Datasets]()” – SPEC ICPE’20

Scaling method

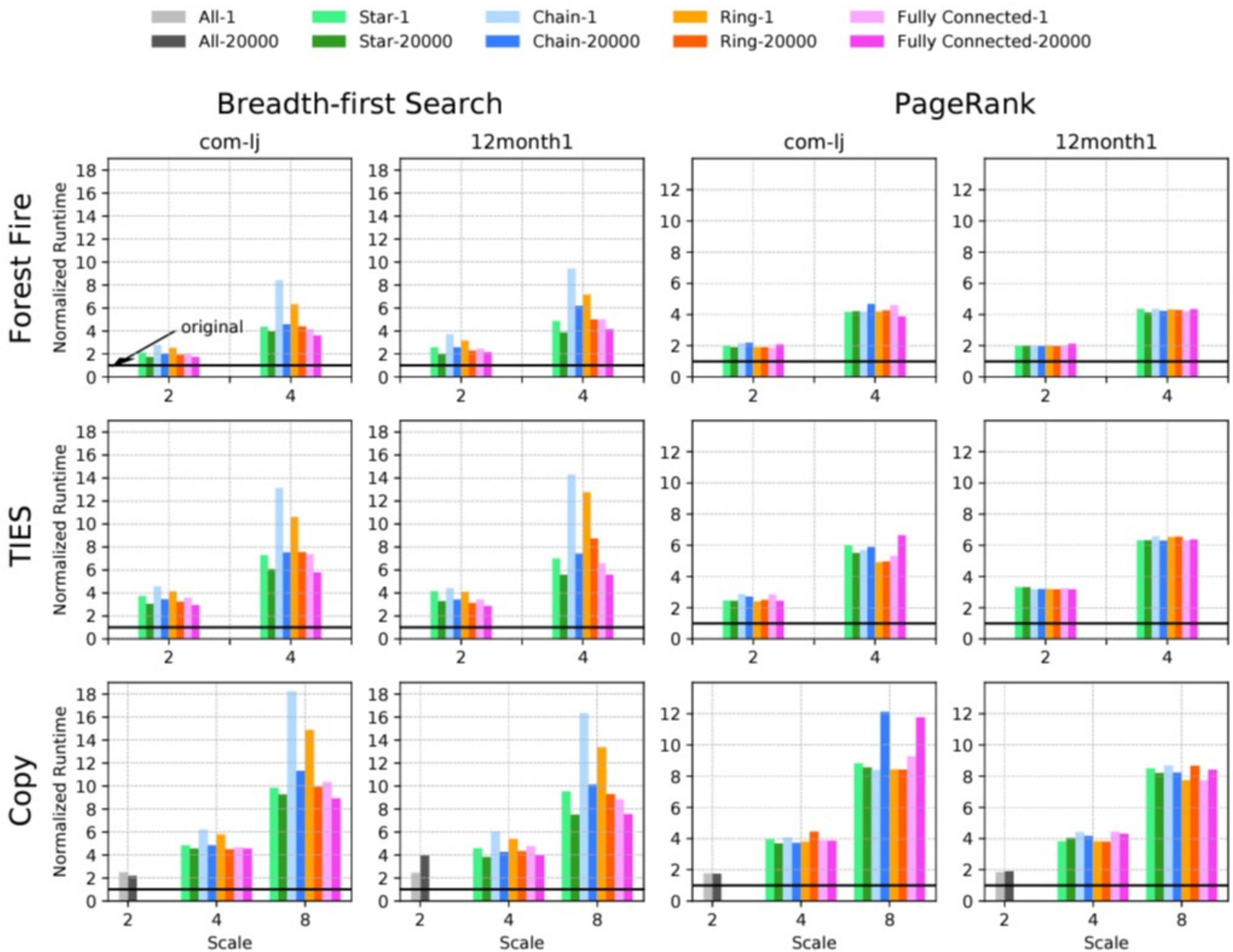
- Scale-down: sampling

- Scale-up: ”stiching”
  - Interconnection
  - Select bridge vertices
  - Multi-edge interconnections



a. star
b. chain
c. ring
d. fully connected

Results



All-1
All-20000
Star-1
Star-20000
Chain-1
Chain-20000
Ring-1
Ring-20000
Fully Connected-1
Fully Connected-20000
Breadth-first Search
PageRank
com-lj
12month1
com-lj
12month1
Forest Fire
TIES
Copy
Normalized Runtime
original
Scale

Lessons learned

- Graph up-/down-scaling can lead to interesting graph families
- Successful controlled experiments
  - Some scalability trends are visible
  - … but not for all graphs
- Performance prediction still quite inaccurate
- Still “needle-in-the-haystack” analysis …

4. Performance prediction

Choose the best algorithm

- Model the algorithm
  - Basic analytical model (work & span)
- Calibrate to platform
  - GPU, CPU, …
- Model the dataset
  - Size, dimension, topology …

$$\mathbf{T = f(P, A, D)}$$

- Predict performance
  - Plug the platform and graph parameters into algorithm model

- Rank solutions and pick best.

Analytical models

Example: PageRank

- Edge-centric

Algorithm 1 Edge List Update & Reverse Edge List Update

---

function $\textsc{EdgeList}(edges, pageranks, new\_pageranks, idx)$
  $origin \leftarrow edges[idx].origin$
  $dest \leftarrow edges[idx].destination$
  $outgoingRank \leftarrow 0$
  if $degree(origin) \neq 0$ **then**
    $outgoingRank \leftarrow \frac{pageranks[origin]}{degree(origin)}$
  $new\_pageranks[dest].atomicAdd(outgoingRank)$

$$T_{\text{edge}} = \sum_{e \in E} (4 * T_{\text{read}} + T_{\text{atom}})$$
$$= 4 * |E| * T_{\text{read}} + |E| * T_{\text{atom}}$$

PageRank: Conceptual analytical models

- Different algorithms => different models

$$T_{\text{edge}} = \sum_{e \in E} (4 * T_{\text{read}} + T_{\text{atom}}) = 4 * |E| * T_{\text{read}} + |E| * T_{\text{atom}}$$

$$T_{\text{push}} = \sum_{v \in V} (2 * T_{\text{read}} + d_v * T_{\text{atom}}) = 2 * |V| * T_{\text{read}} + |E| * T_{\text{atom}}$$

$$T_{\text{pull}} = \sum_{v \in V} (2 * d_v * T_{\text{read}} + T_{\text{write}}) = 2 * |E| * T_{\text{read}} + |V| * T_{\text{write}}$$

- Extracted from the algorithms' ps
  - Not accurate enough, as there are m
    practice …

PageRank: Conceptual analytical models

- Different algorithms => different models

$$T_{\text{edge}} = \sum_{e \in E} (4 * T_{\text{read}} + T_{\text{atom}}) = 4 * |E| * T_{\text{read}} + |E| * T_{\text{atom}}$$

$$T_{\text{push}} = \sum_{v \in V} (2 * T_{\text{read}} + d_v * T_{\text{atom}}) = 2 * |V| * T_{\text{read}} + |E| * T_{\text{atom}}$$

$$T_{\text{pull}} = \sum_{v \in V} (2 * d_v * T_{\text{read}} + T_{\text{write}}) = 2 * |E| * T_{\text{read}} + |V| * T_{\text{write}}$$

Extracted from the algorithms' **pseudocode.**
Not accurate enough, as there are more operations executed in practice …

PageRank: Code-based analytical models

- Different algorithms => different models

$$
\begin{aligned}
T_{\text{edge}} &= (7 * |E| * T_{\text{read}} + |E| * T_{\text{atom}}) \\
&\quad + (2 * |V| * T_{\text{read}} + 2 * |V| * T_{\text{write}} + \frac{|V|}{32} * T_{\text{atom}}) \\
&= (7 * |E| + 2 * |V|) * T_{\text{read}} + 2 * |V| * T_{\text{write}} + (|E| + \frac{|V|}{32}) * T_{\text{atom}}
\end{aligned}
$$

$$
\begin{aligned}
T_{\text{push}} &= (6 * |V| * T_{\text{read}} + |E| * T_{\text{read}} + |E| * T_{\text{atom}}) \\
&\quad + (2 * |V| * T_{\text{read}} + 2 * |V| * T_{\text{write}} + \frac{|V|}{32} * T_{\text{atom}}) \\
&= (8 * |V| + |E|) * T_{\text{read}} + 2 * |V| * T_{\text{write}} + (|E| + \frac{|V|}{32}) * T_{\text{atom}}
\end{aligned}
$$

$$
\begin{aligned}
T_{\text{pull}} &= (5 * |V| * T_{\text{read}} + 3 * |E| * T_{\text{read}} + |V| * T_{\text{write}}) \\
&\quad + (2 * |V| * T_{\text{read}} + 2 * |V| * T_{\text{write}} + \frac{|V|}{32} * T_{\text{atom}})
\end{aligned}
$$

$$V| * T_{\text{write}} + \frac{|V|}{32} * T_{\text{atom}}$$

Based on PTX (that is, NVIDIA GPUs' assembly)
Calibrate for the platform : T_{read}, T_{write}, T_{atom} …
Use dataset features: |E| and |V| from the graph specs

Validate models

- Work-models are correct
  - We capture correctly the number of operations
- Model calibration has failed
  - Workload imbalance between threads within a warp
  - Non-uniform memory access times due to coalescing, caching, and atomic contention.
- Can we do any better?
  - Model parallelism to better understand the variation in T's
  - Use performance counters to capture different aspects of T's

Machine learning to the rescue?

Choose the best algorithm

- Model the algorithm
  - Basic analytical model (work & span)
- Calibrate to platform
  - GPU, CPU, …
- Model the dataset
  - Size, dimension, topology …

T = f(P, A, D)

- Predict performance
  - Plug the platform and graph parameters into algorithm model

Only 50% accuracy ☹

- Rank solutions and pick best.

Data and models

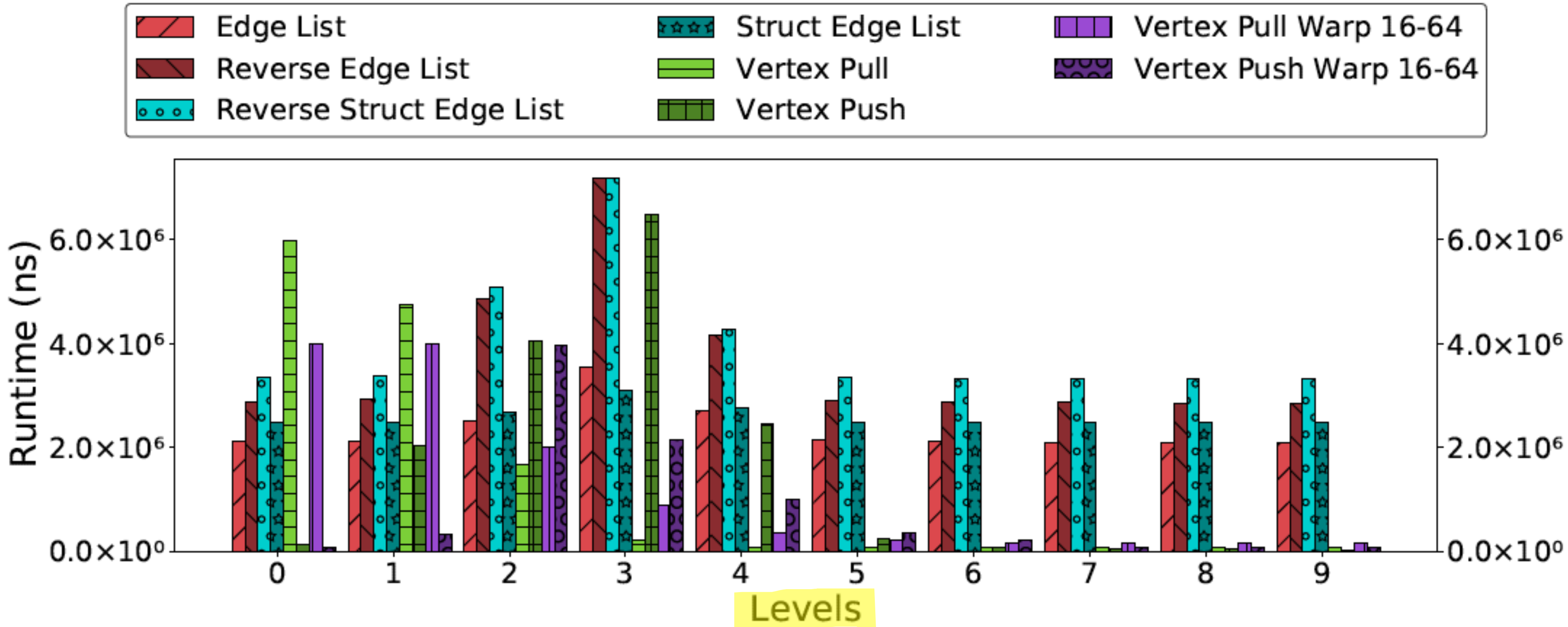
- Build dataset from 200+ graphs and ~20 different roots
- Collect performance data from different platforms and algorithms

- Devise models to…
  - Predict execution time
    - Use random forest
      - Based on hardware counters (previous work)
      - Based on graph features
  - Predict ranking
    - Use decision trees
      - Based on graph features

PageRank
Reasonable accuracy,
High prediction cost

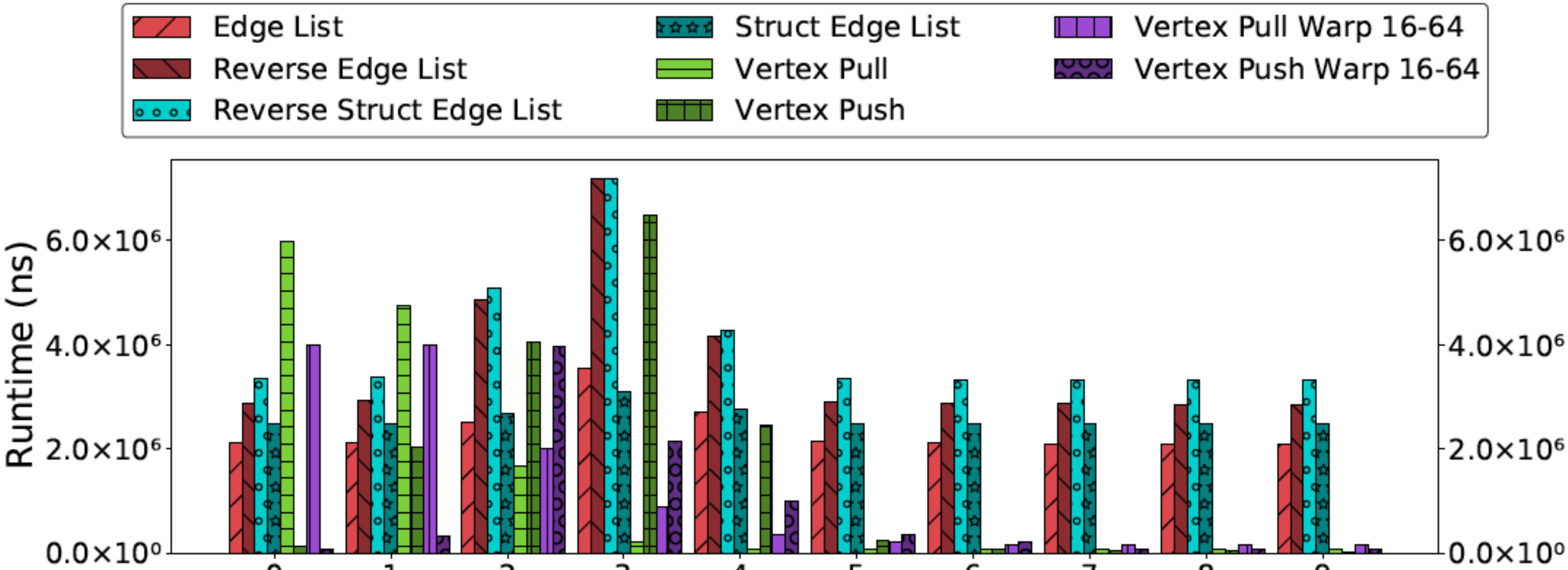
PageRank
High accuracy,
Low prediction cost
Still not working for BFS!!!

BFS: best algorithm changes!



Results on the different BFS levels for the actor-collaborations graph (ID #1)

BFS: best algorithm changes!



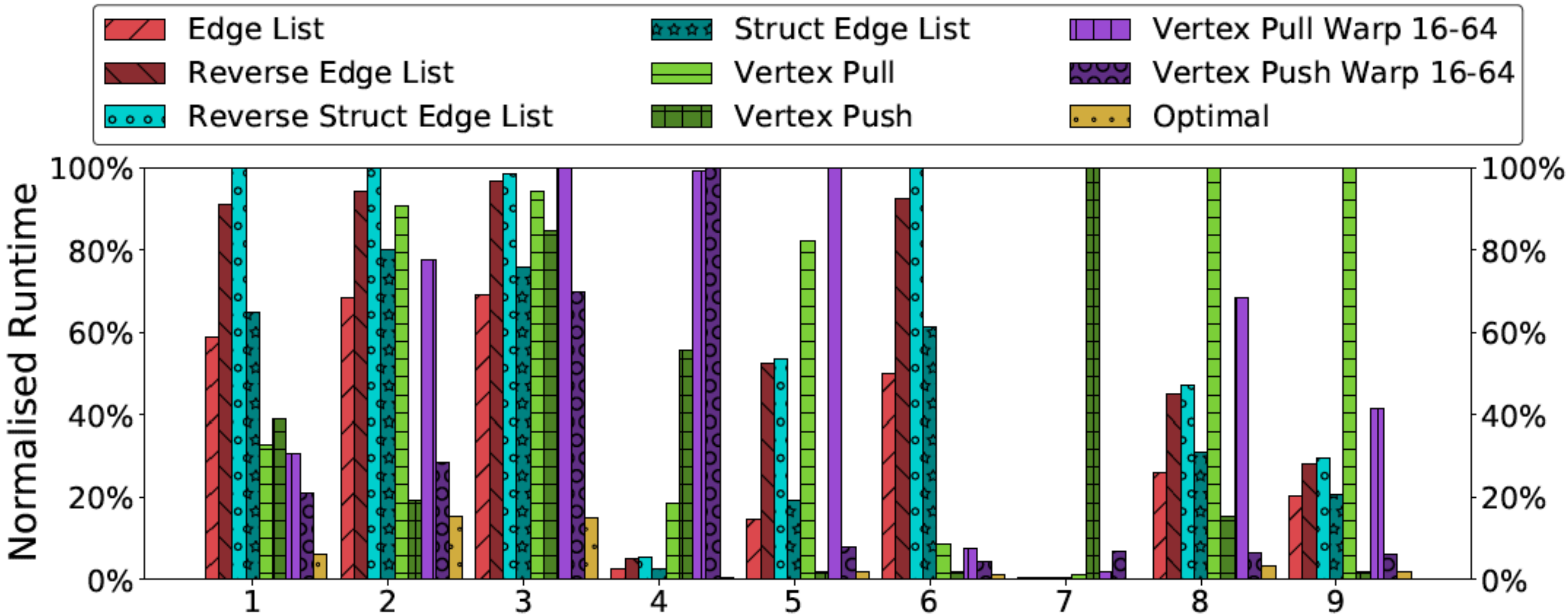
- Best algorithm changes per level
- Gaps are even larger than for the full scale
- We have more data for every level

We must predict at every level, NOT at the full graph level !

Results on the different BFS levels for the actor-collaborations graph (ID #1)

5. The best BFS algorihm

BFS: *construct* the best algorithm!



- Optimal algorithm is the sum of the best per-level algorithms.
- Must switch implementations

If we predict best algorithm per level => we construct the best algorithm

BFS: *construct* the best algorithm!

- Predict ranking
  - Determine the **best algorithm per level**
  - **Still** depends on platform and dataset …
- **Construct** the best overall algorithm
  - Best algorithm per layer => best overall *by construction*
  - Switching between algorithms is a challenge
    - When?
    - How?

Mix-and-match: build the best algorithm at run-time by **switching to the best implementation** at every level*

*this is a generalization of the direction-switching BFS

Predicting ranking per level

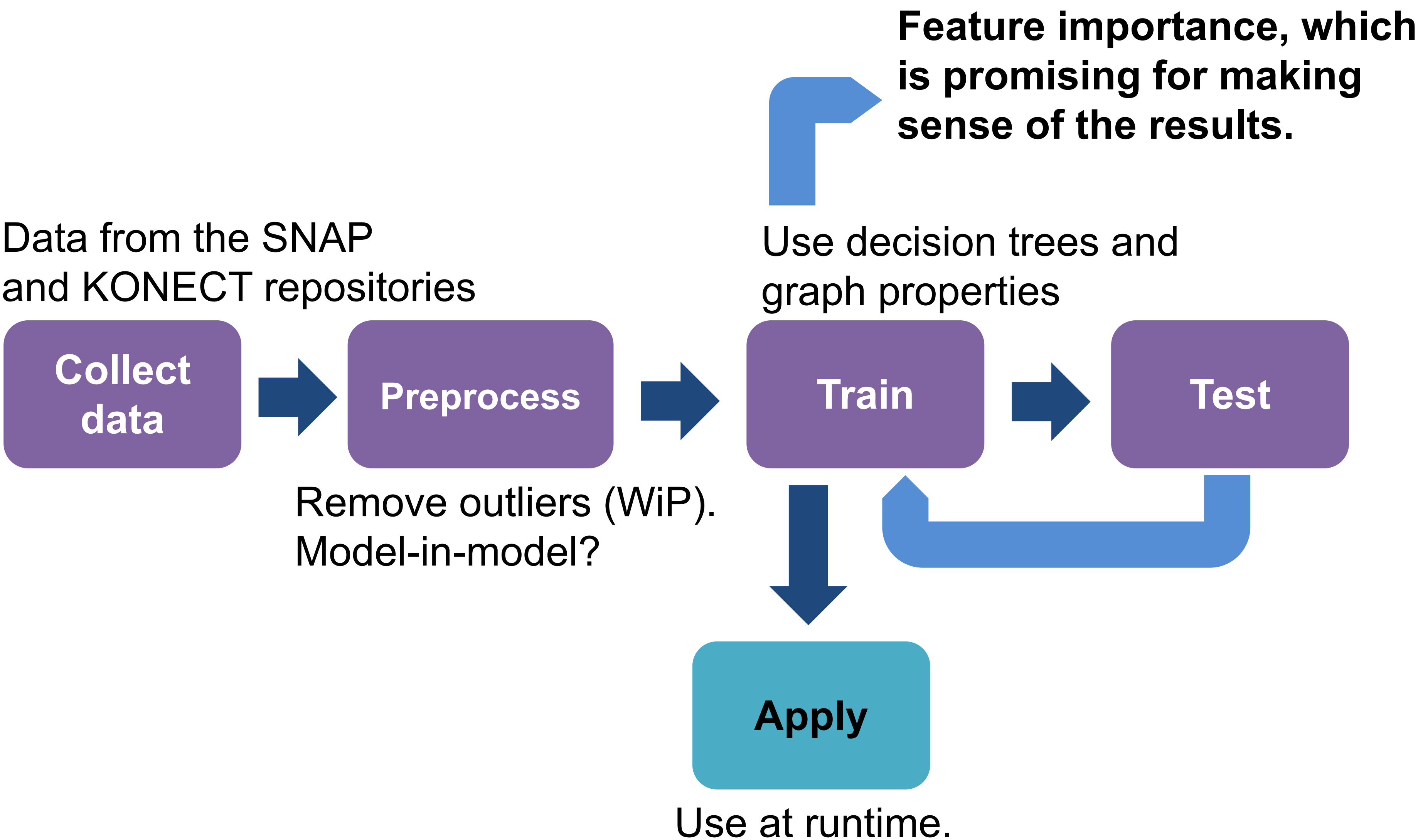
- Based on decision trees
  - Small number of samples
  - Fairly easy to train
  - Model is fast to use at runtime

Average prediction time: 144ns
Min BFS step: 20ms

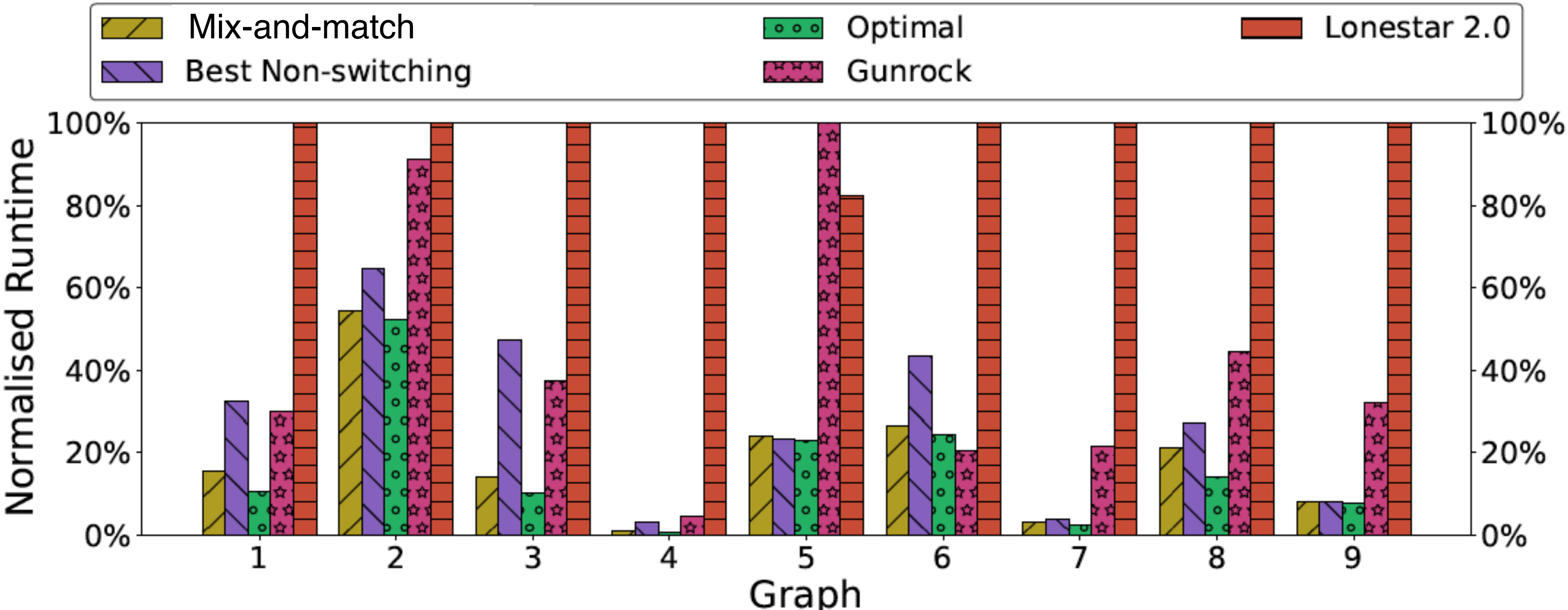
- Training parameters: **graph features** and **best algorithm**
  - Degree distribution (5 number summary and standard deviation)
  - *Frontier size*
  - *Percentage discovered*
  - Vertex count
  - Edge count
  - Ranking

Dataset: 248 graphs x ~11 root nodes
Accuracy: ~98%

Current workflow



Does it really work?



- Runtime switching is possible, (currently) with some memory overhead
- We are faster than the state-of-the art, on average, by 3x

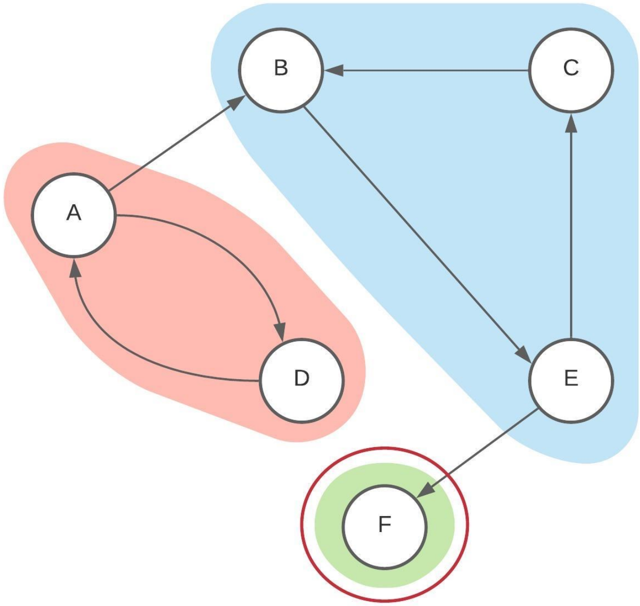
Mix-and-match uses performance variability to build the best BFS per graph!

6. Improving SCC

Detecting strongly connected components*

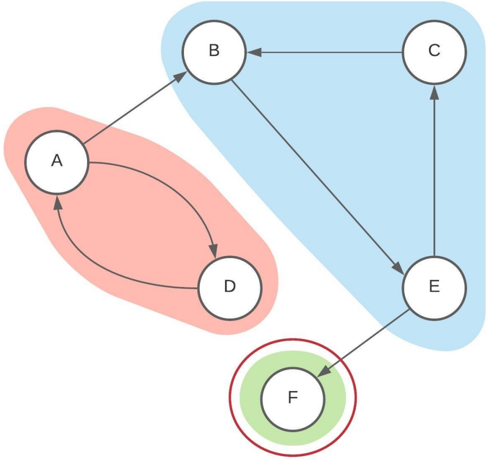
- SCC
  - Subgraph of a directed graph
  - Every vertex *u* can reach every other vertex *v* in the subgraph.

- Used in applications such as:
  - Community detection
  - Personalized recommendations
  - Program analysis



*Dante Niewenhuis, Ana-Lucia Varbanescu
Efficient Trimming for SCC Calculation, CF'22

FB-Trim

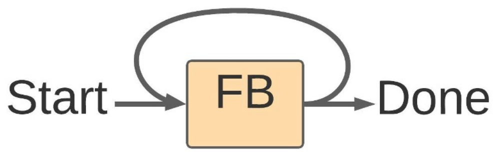


- FB = Forward-Backward algorithm
  - First **parallel** SCC algorithm, proposed in 2001
    - The base for most parallel SCC algorithms
- Problem: **Trivial components**
  - Trivial components consist of only a single vertex (e.g., F)
- Solution? **Trimming**
  - *Iteratively* remove floating vertices
- **FB-Trim** Combines FB and Trimming …
- … but its performance is **dependent on graph topology**.

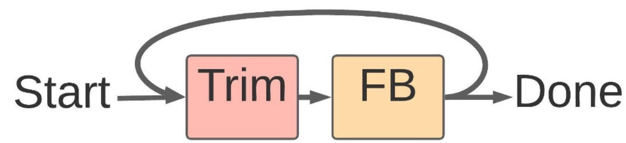
When should we trim for a good trade-off between effectiveness and overhead?

Static trimming models

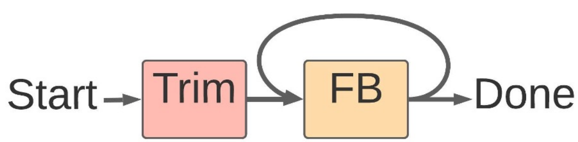
Never Trim



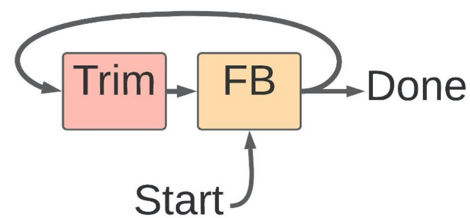
Always Trim



Initial Trim



No Initial Trim



Experimental setup and method

- Data : 819 graphs
  - KONECT and Network Repository
  - Graph Size: 500 - 10.000 vertices
- Per graph: Measure execution time (6x) and report average
  - Execution time is capped at 5 minutes.
- Platform:
  - Lisa* node: Intel Xeon Silver 4110 Processor at 2.1GHz, 96 GB RAM
  - Software: Ubuntu 20.04, C++ 14 compiled with GCC 9.3.0, Python 3.7.6

*https://userinfo.surfsara.nl/systems/lisa/description

Aggregated results analysis

- Ranking-based
  - Best and Worst
  - Average Ranking

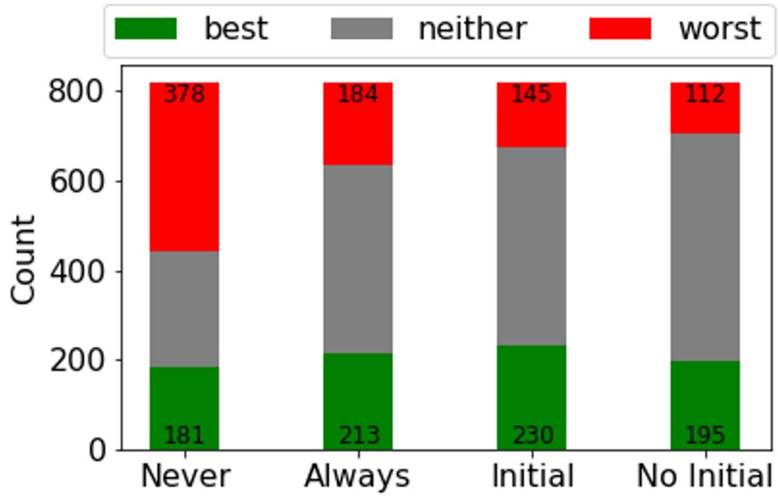
- Time-based
  - Average execution time over all the graphs
  - Relative Increase*:

$$RI(G_i, M_k) = (T(G_i, M_k) - T_B(G_i))/T_B(G_i) * 100$$

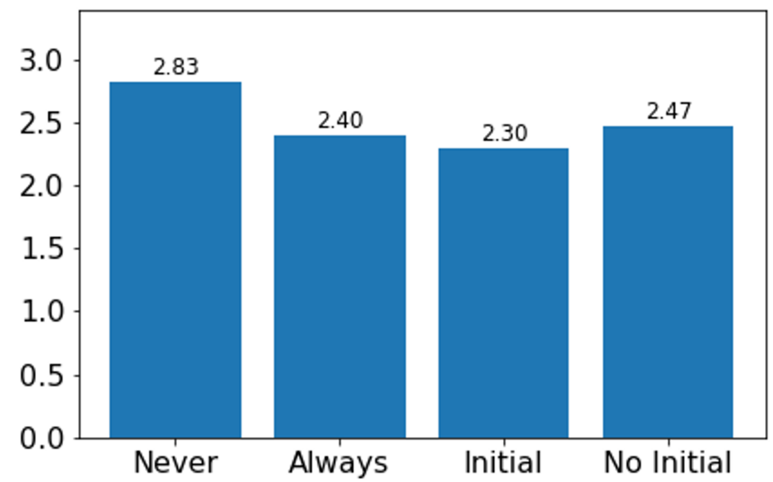
*where G_i is graph *i*, M_k is model *k*, $T(G_i, M_k)$ is the execution time of model *k* on graph *i*, and $T_B(G_i)$ is the best time on graph *i*

The static models' performance [1/2]

Best and Worst placement

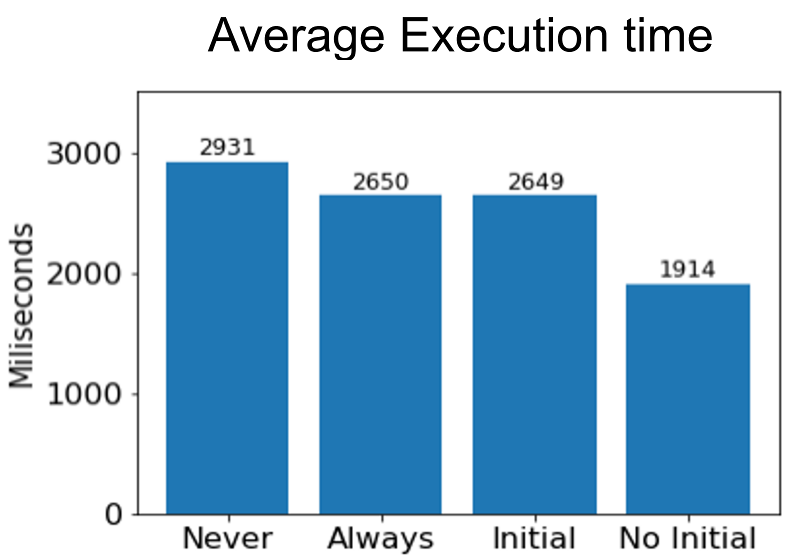


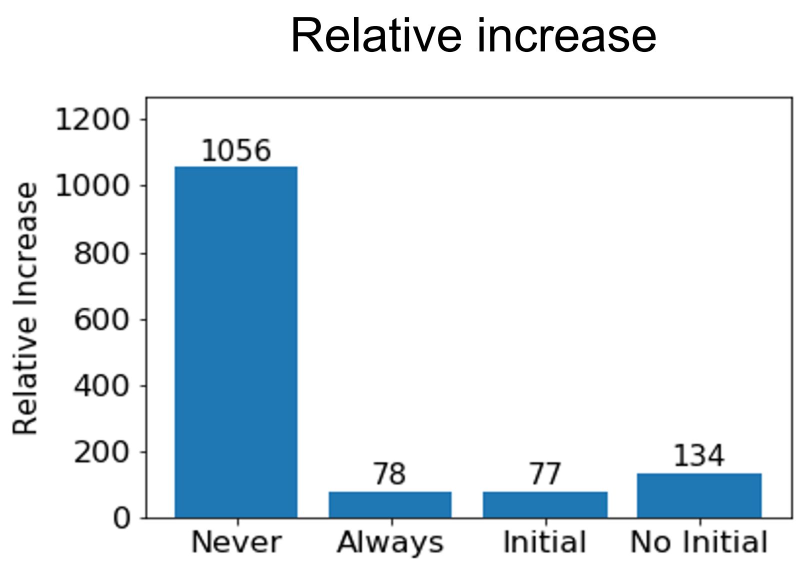
Average Ranking



- Each model performs best on at least 15% of the graphs
- Each model performs worst on at least 15% of the graphs
- **None** of the models significantly outperforms all other models in ranking

The static models' performance [2/2]



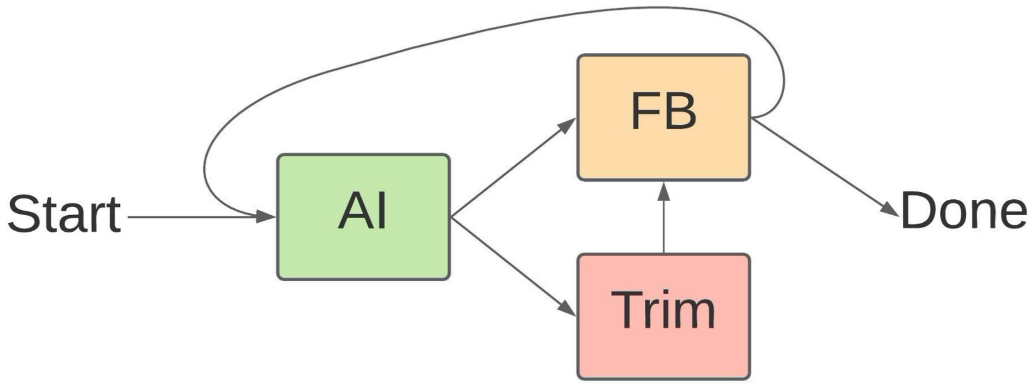


- **Average execution time** indicates **No-Initial** is the best.
- *Never Trimming* performs horribly on the average RI.
- Best performing model has an average RI of over 77%.

Problem: no static model outperforms all other models consistently.

Predict trimming efficiency using AI

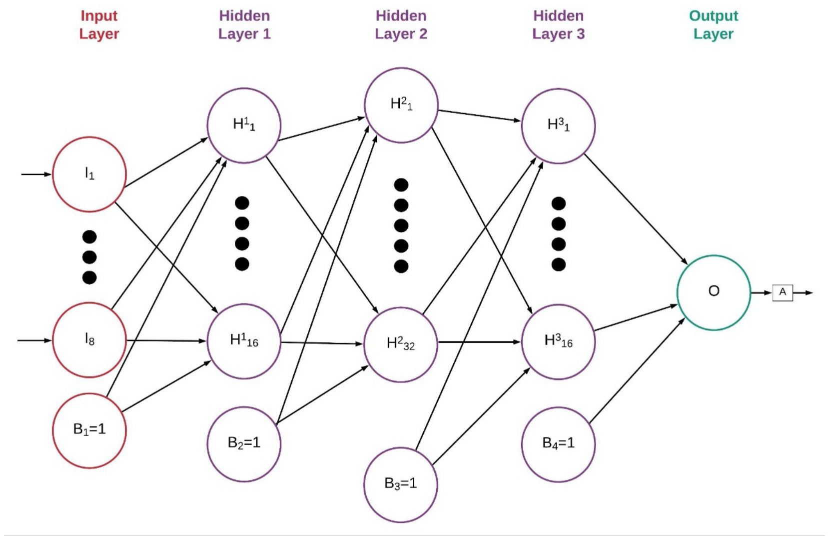
- A NN-based model that determines when to trim based on graph topology



The AI model

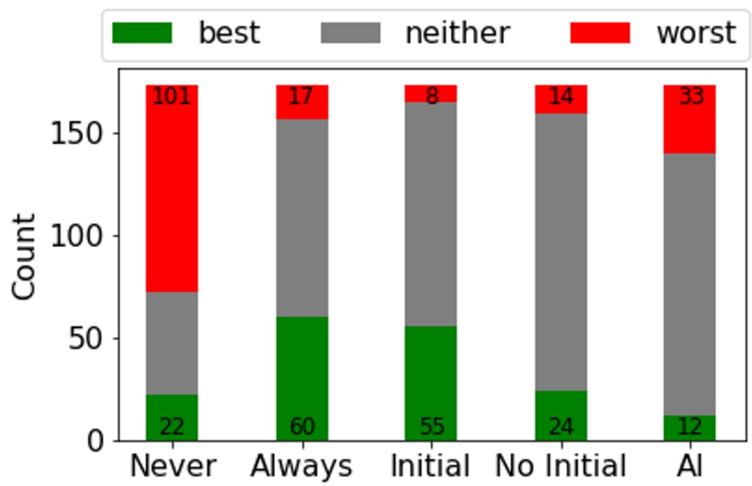
- Basic Neural Network
  - Input layer length 8
  - Three hidden layers
  - Boolean output layer

- Training
  - Using Gradient Descent
  - For 5000 epochs
  - Reached an accuracy of 82%

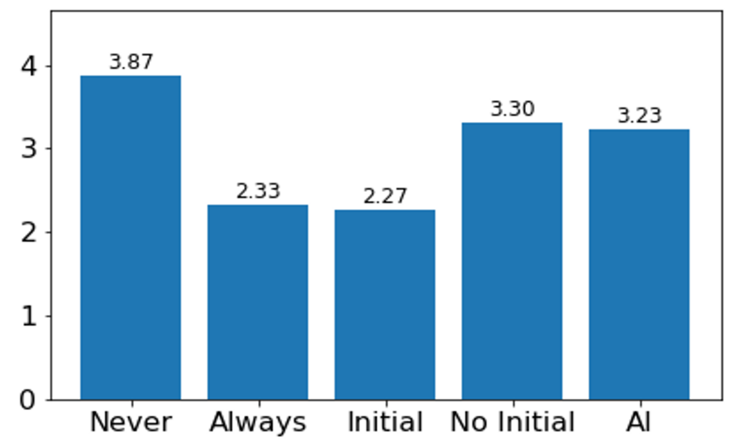


The AI model's performance [1/2]

Best and Worst placement



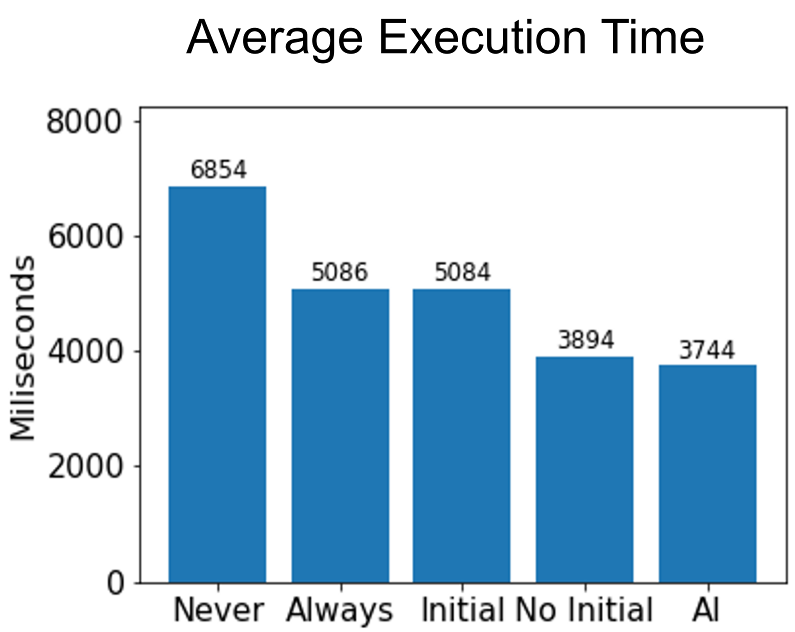
Average Ranking

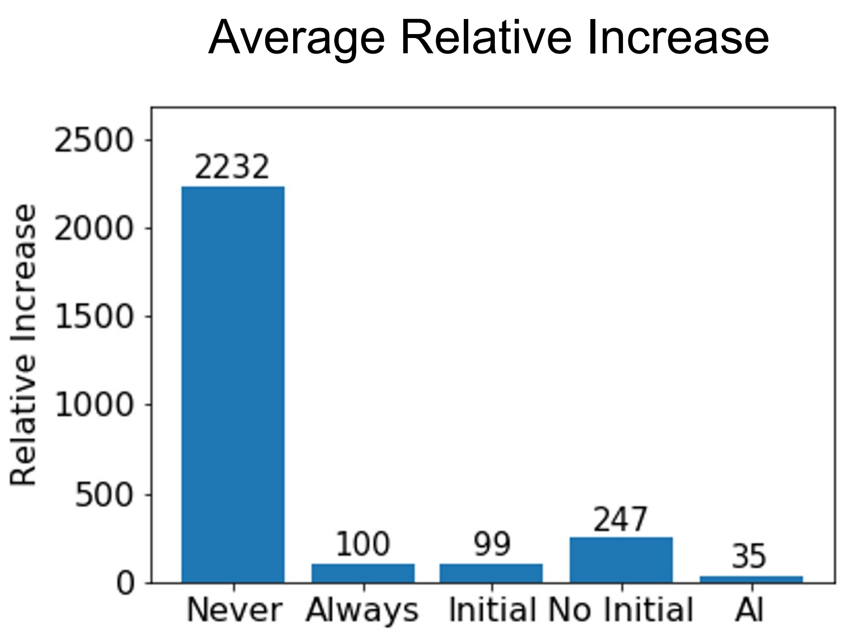


AI-Trim performs poorly in single graph based metrics:

- lowest number of best-performing graphs.
- second highest number of worst-performing graphs, after Never Trim.
- Mid-of-the-pack average ranking.

The AI model's performance [2/2]



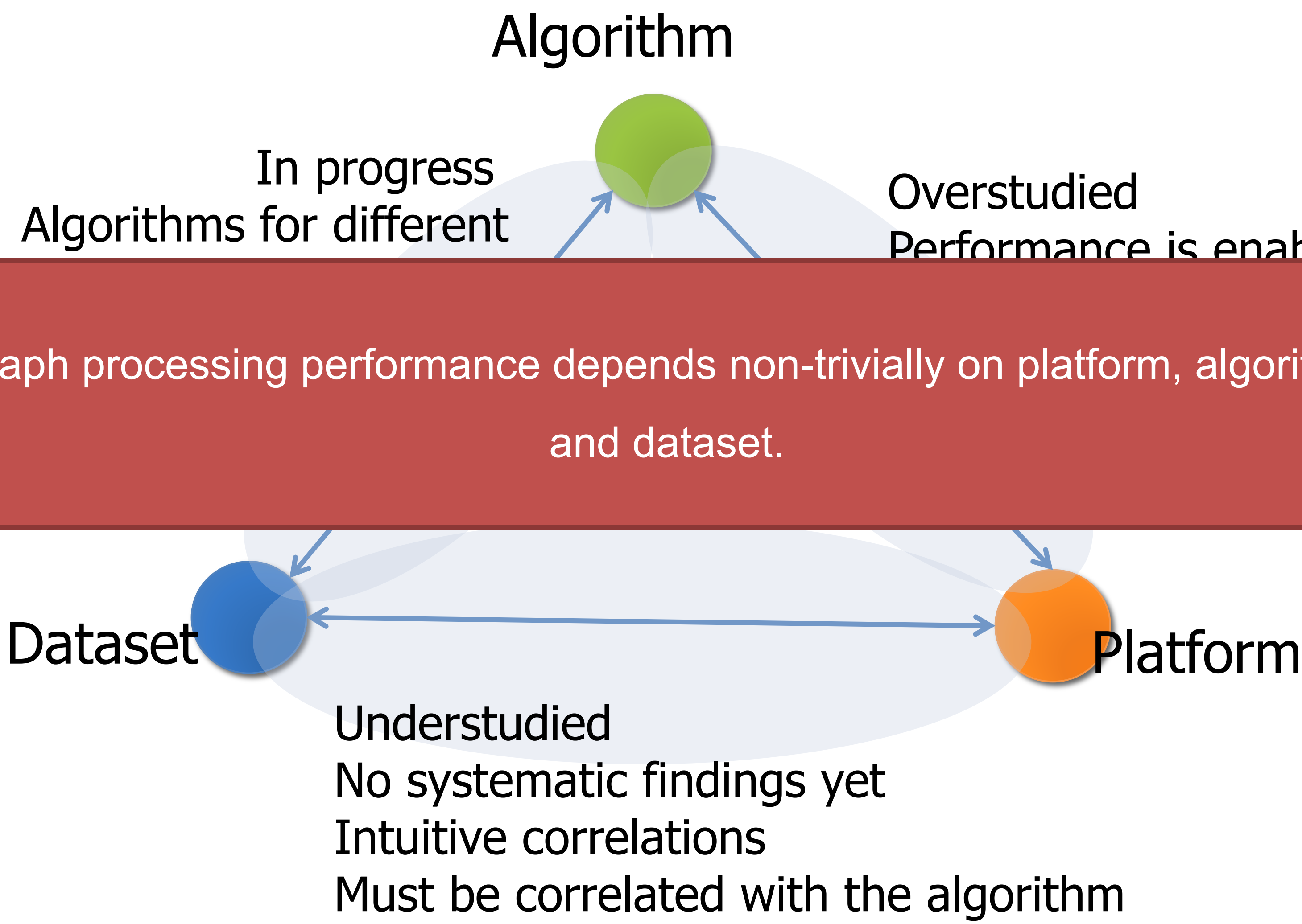


AI-Trim **outperforms** all other models on execution time-based metrics:

- **lowest average execution** time of all models.
- RI = 35% is almost 3x better than the next best model.

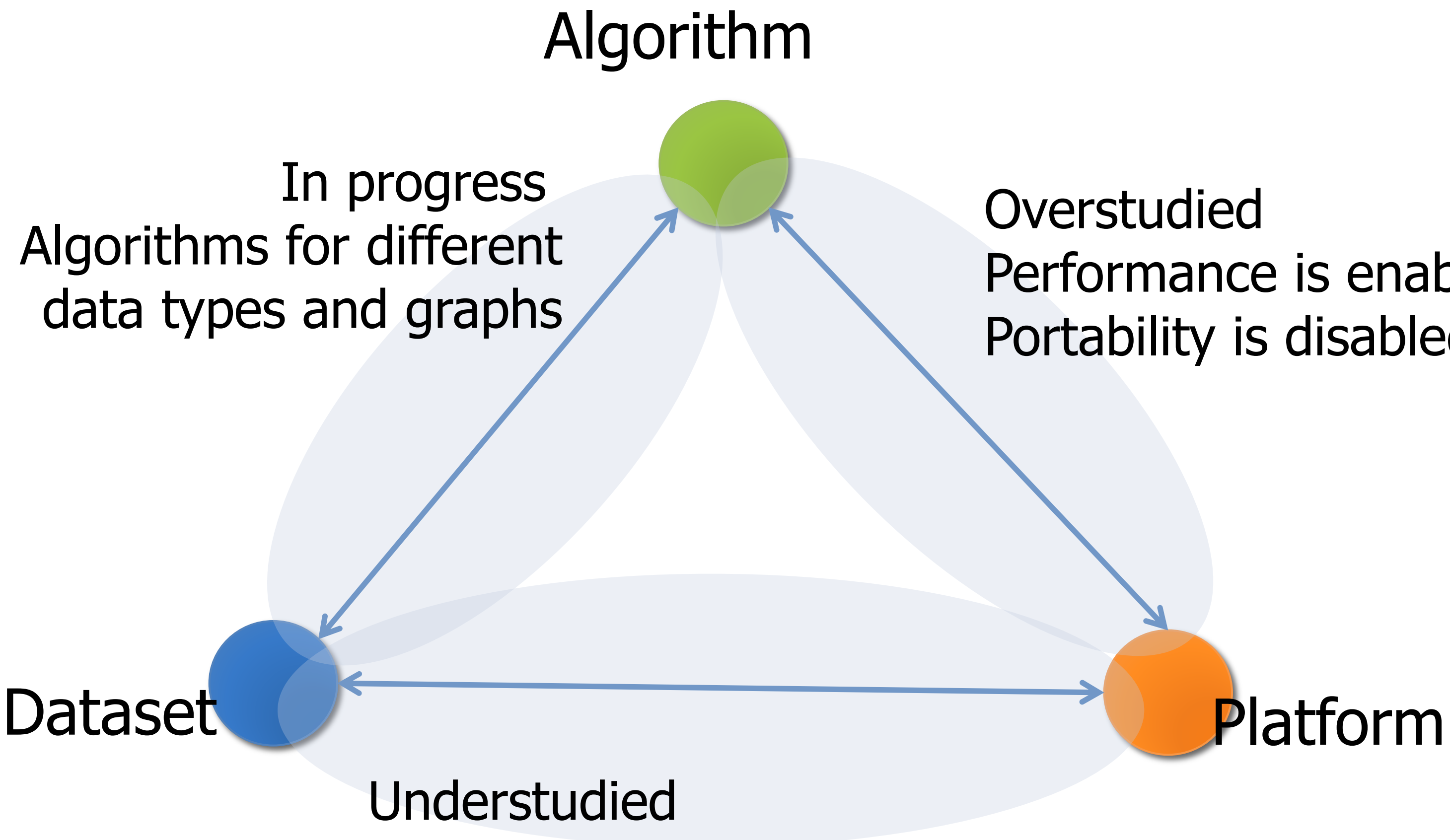
7. Take home message

P-A-D triangle



Graph processing performance depends non-trivially on platform, algorithm, and dataset.

P-A-D triangle



In progress
Algorithms for different
data types and graphs

Overstudied
Performance is enabled
Portability is disabled

Understudied
No systematic findings yet
Intuitive correlations
Must be correlated with the algorithm

Take home message

- Graph scaler offers graph scaling for controlled experiments
  - Correlation between performance and graph features is still WiP
- Mix-and-match creates best BFS by enabling dynamic, runtime switching among different versions of BFS
  - A generalization of the direction-optimized BFS
  - Machine learning model used to guide the switching
- FB-Trim improves the efficiency of trimming using a simple NN model to determine when to trim
  - Decision made on the graph topology … we think

Ana:	A.L.Varbanescu@utwente.nl
Merijn:	merijn@inconsistent.nl
Dante:	d.niewenhuis@hotmail.com
Graph scaling:	https://github.com/amusaafir/graph-scaling
M&M:	https://github.com/merijn/Belewitte
FB_Trim:	https://github.com/DanteNiewenhuis/FB-Trim